Project2

November 6, 2020

1 Project 2

1.1 Imports

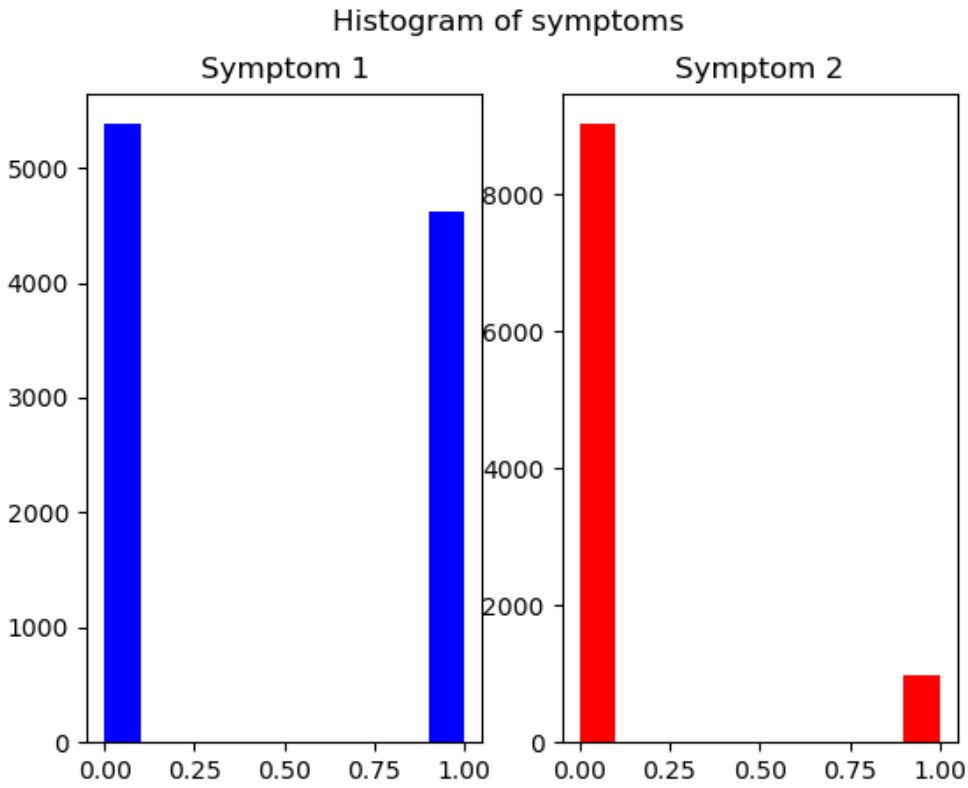
[2]:
```
%matplotlib inline
from IPython.display import Image
from IPython.core.display import display
```

1.2 Historical data

1.2.1 Structures in the data

Overview of the data We could look at the simple distribution of the two symptoms. This shows the frequency of the two symptoms in the entire dataset. From the histogram it could seem like symptom 2 is less frequent than symptom 1.

[3]:
```
display(Image(filename="img/freq_hist.png", width = 600))
```



Single cause versus multiple cause model Just by looking at the simple histogram of frequencies we see that the distribution of the two symptoms are very different. Symptom 2 seems to be far less frequent than symptom 1. In order to look further into into unknown clusters in the data, we could use unsupervised learning techniques such as K-means clustering or hierarchical clustering. Our overall goal is to predict the symptoms based on the covariates which are gender, smoker and gene expression data. Because of the number of covariates (the number of different gene expressions), it is not trivial to examine if there is a single cause (covariate) for the symptoms or not.

There could also be a subset of gene expression that are the main cause of the symptoms, but determining which of the gene expression that explains the most of the variability in the symptoms could be more difficult to infer. This is closely related to the concept of confounding covariates. Two covariates, if correlated, could be subject to confounding in such a way that the effect of the second covariate could be partly expressed through the first covariate. In the case of two covariates, the second is then affecting through $\hat{\rho}\sqrt{\frac{\sigma_1}{\sigma_2}}$ where $\hat{\rho}$ is the correlation between the covariates, σ_i is the standard deviation of the covariate i (Borgan, 2019, p. 1). In our case this could be the case if we include some number of genes as covariates. Some genes could be confounded by other genes, masking their influence on the response variable.

By attempting to narrow down the covariates that cause a particular symptom, we are looking at a "causes of effect" problem (Dawid, 2015, p. 5). That is, what covariates causes the symptoms

observed in the data set. When it comes to data sets that we observe after the "experiment" has taken place (that is, collecting the data) there should ideally be exchangeability among the observations (Dawid, 2015, p. 21). This condition may be difficult to verify when looking at the dataset, especially when taking into account the uncertainty that (may) be involved when registering data for the symptoms. These could range from objective measurese such as a level of concentration or to more subjective measures such as an individual assessment of symptoms.

Further, Dawid also discusses the negative impact of confounding when observing the data. In order to achieve this, there must in principal be exchangeability between groups within the data. He then considers the case of having the treated groups being exchangeable with the non-treated group in order to have exchangeability and that this in theory happens when the observations are randomised (Dawid, 2015, pp. 21-22). Dawid then relates the absence of confounding to independence between the covariates (Dawid, 2015, p. 22). This can also be considered when looking at the definition from Borgan above, where confounding is expressed through correlation $\hat{\rho}$ between the covariates.

K-means clustering The idea behind K-means clustering is to find the clusters that minimize the variation within each of the clusters. The distance between two observations could then be measured as the squared euclidean distance between the C covariates of the two observations

$$\sum_{j=1}^{C}(x_{ij} - x_{i'j})^2$$

adapted from (James et al., 2013, p. 387). Further, we then want to minimize the following equation in order to obtain the "optimal" clusters where G_k is one of the K clusters and we sum the squared euclidean distance between points within the same cluster G_k

$$\sum_{k=1}^{K}\frac{1}{|G_k|}\sum_{i\cap i'\in G_k}\sum_{j=1}^{C}(x_{ij} - x_{i'j})^2$$

adapted from (James et al., 2013, p. 387).

The algorithm then computes the "mean" observation for each cluster which in fact is the mean of all the covariates within the cluster. Then the observations are placed in the clusters in which they are closest to measured in for example squared euclidean distance (James et al., 2013, p. 388).

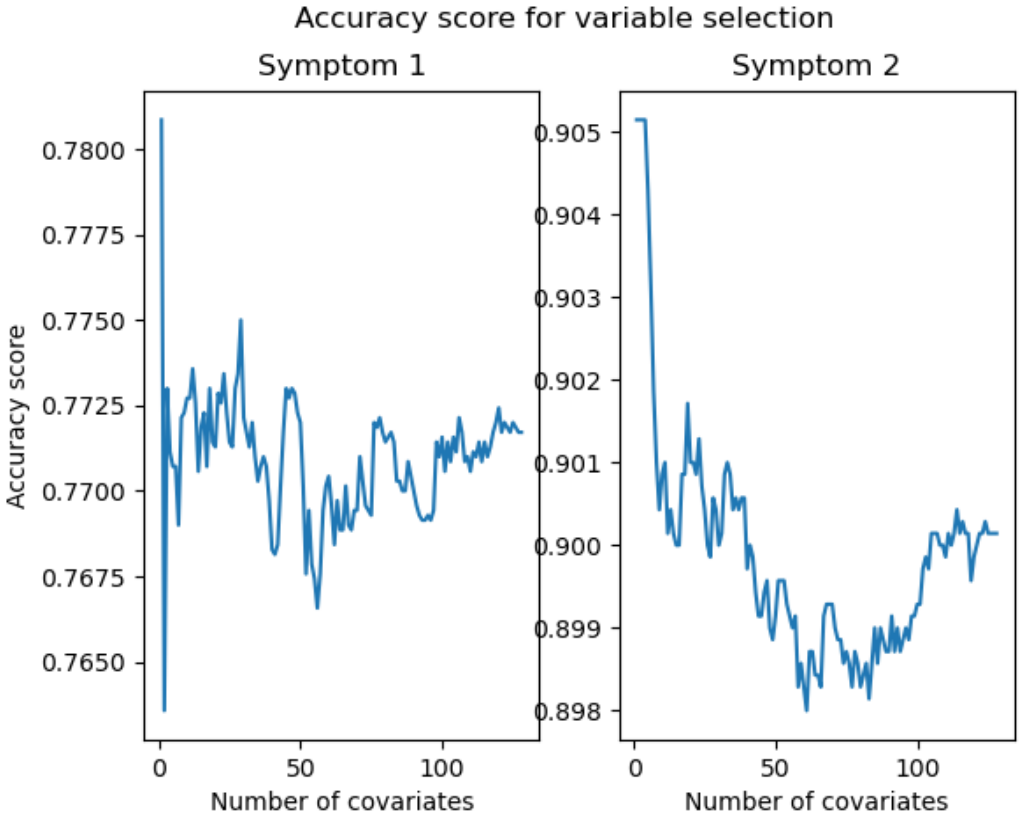
In this example, we could use this algorithm with $K = 2$ and check the clusters created by the algorithm with the different symptoms. Intuitively, if a large percentage of the cluster 1 observations exhibit one of the symptoms compared to those observations in cluster 2 it could be reason to investigate closer if there in fact are two different clusters of observations, each mainly related to each of the symptoms. Unfortunately, the K-means clustering will be difficult to perform due to the binary structure of the data.

Logistic regression I will instead look into logistic regression as a model for the data. There could be defined a model for each of the covariates and all combinations of covariates up to the full model with all the covariates. First, I want to try using cross-validation on a valdiation set (within the training data) in order to select the covariates that make up the "best" model when it comes

to prediction. In that way, I could say that it is more likely that there is a single-cause model if the best model w.r.t. prediction is a model with a single covariate. On the other hand, if the number of "optimally" chosen covariates are > 1 then this could imply that there is in fact a multiple-cause model that is more likely. It would also have been possible to use different forms of information criterias in order to perform variable selection, such as AIC or BIC. This would have penalized the number of covariates directly to avoid overfitting. An alternative to this is cross-validation on the training data where the cross-validation error is the misclassification rate. This is then measured on the validation set as described below.

In order to check the effect of the different covariates, I used recursive feature elimination with cross-validation in order to select the covariates based on the misclassification score (scikit learn, 2020). This is then specified to use a stratified K-fold cross-validation (because of the few occurences of symptom 2). What is interesting is the fact that for both symptoms, the variable selection selects one covariate for each symptom, 'gene 4' for symptom 1 and 'gene 3' for symptom 2 based on 3-fold cross-validation on the training data of the dataset. Looking at the accuracy score for the logistic regression of the two symptoms

[4]:
```
display(Image(filename="img/var_sel.png", width = 600))
```



Hierarchical model The relationship we are interested in is given by $y_t|a_t$, that is, how is the outcome for the observation dependent on the intervention (a_t). Before we can do this, we would

like to consider the causes for the symptoms x_{129} and x_{130}. This can be considered as a hierachical model in the Bayesian framework. If we look at a model posterior, we have

$$\phi(\mu_i|\vec{y}) = \frac{P(\vec{y}|\mu_i)\phi(\mu_i)}{\sum_i P(\vec{y}|\mu_i)\phi(\mu_i)}$$

for a model μ_i, adapted from (Dimitrakakis, 2020, p. 98). The problem we then have is to specify a prior for the different models, this could for example be a noninformative prior (e.g. uniform) if we have no specific information to suggest otherwise prior to selecting the models. For simplicity, we define the single-cause models as having one covariate and the multiple-cause model as having all the covariates. In our case, we then have 129 different models for each of the symptoms. If we consider the different models as being different logistic regression using the different covariates we could define the structure

$$\eta_t(\vec{x}_t) = \beta_0 + \vec{\beta}^T\vec{x}$$

further, this would give the conditional probability

$$Pr(ys_t = 1|x = x_t) = \frac{e^{\eta_t(\vec{x}_t)}}{1 + e^{\eta_t(\vec{x}_t)}}$$

adapted from (Hastie et al., 2016, p. 119). Where ys_t is the symptom response for observation t. After calculating the model, we would then have $P(\vec{y}|\mu_i)$, the likelihood of the observed data given the model μ_i. Obviously, we do not know $\phi(\mu_i)$, so we use a subjective probability distribution ξ (Dimitrakakis, 2020, p. 108) as an estimator of the true (unknown) ϕ.

The symptom response ys_t is then Bernoulli distributed with parameter $\eta_t(\vec{x}_t)$. If we call

$$Pr(ys_t = 1|x = x_t) = p_t$$

the likelihood can then be written

$$l(\vec{\beta}) = \prod_{i=1}^{n} p_t^{ys_t} \cdot (1 - p_t)^{1-ys_t}$$

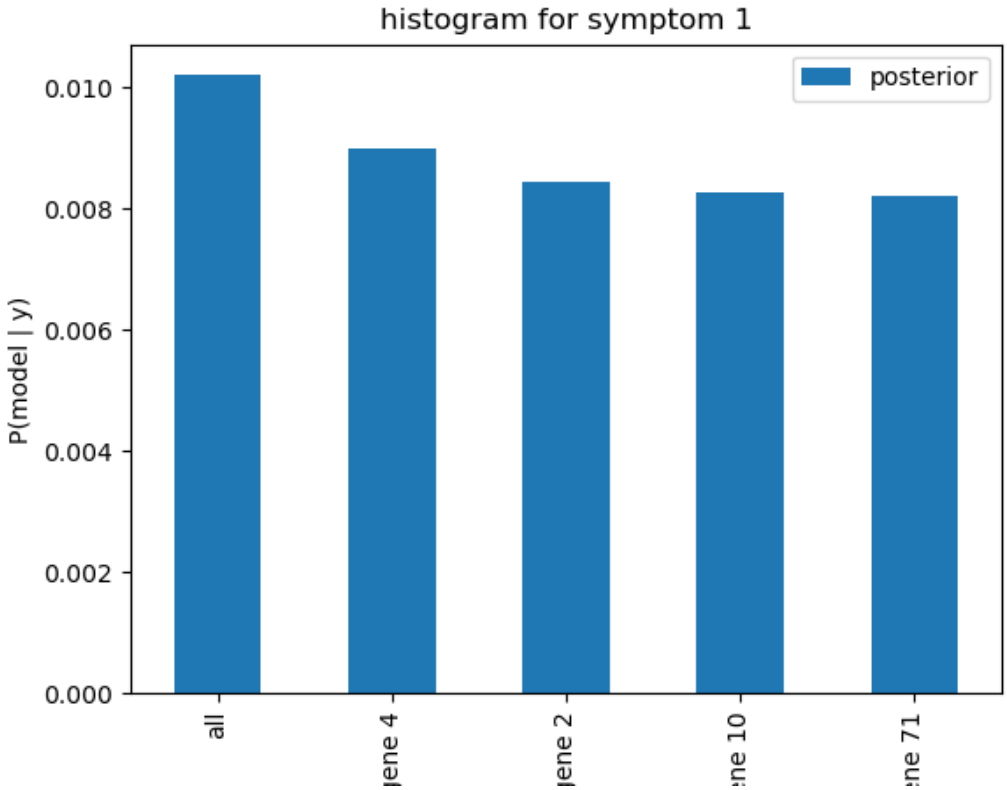
which has log-likelihood

$$loglik(\vec{\beta}) = \sum_{i=1}^{n} ys_t log(p_t) + (1 - ys_t) log(1 - p_t)$$

adapted from (Hastie et al., 2016, p. 120). This will provide us with the opportunity of calculating the likelihood of the data given the different models which then again will allow us to calculate the posterior probability of the different models.

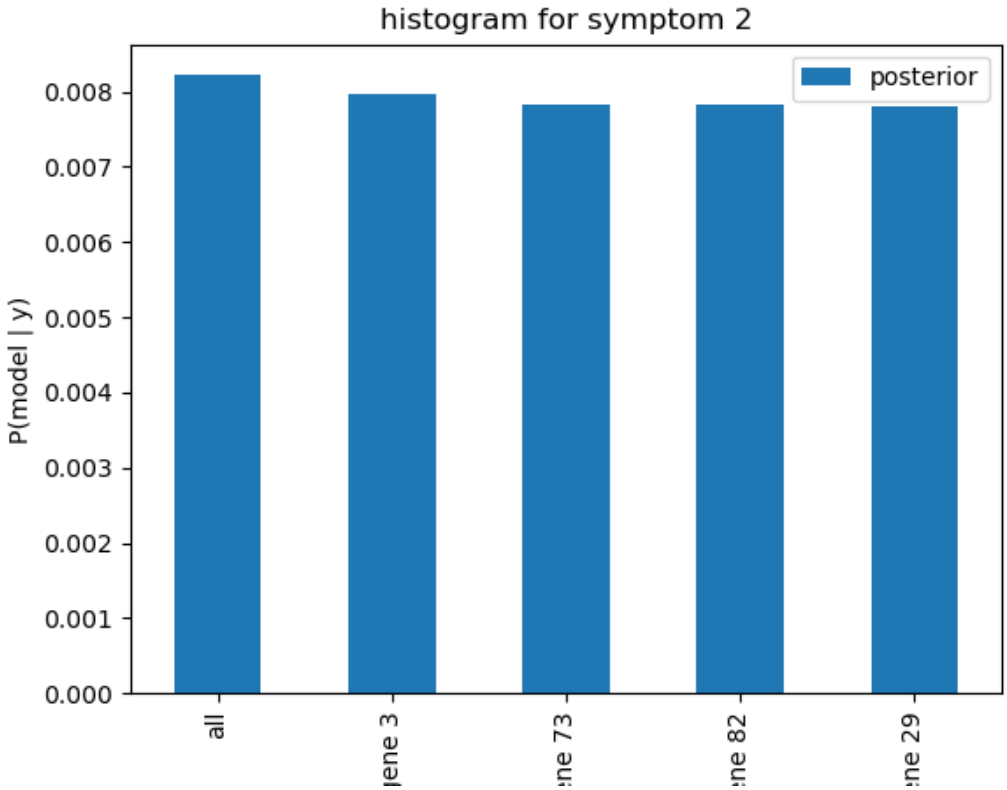
Plotting the posterior probabilities for symptom 1

[16]:
```
display(Image(filename="img/histogram_for_symptom_1.png", width = 600))
```



Plotting the posterior probabilities for symptom 2

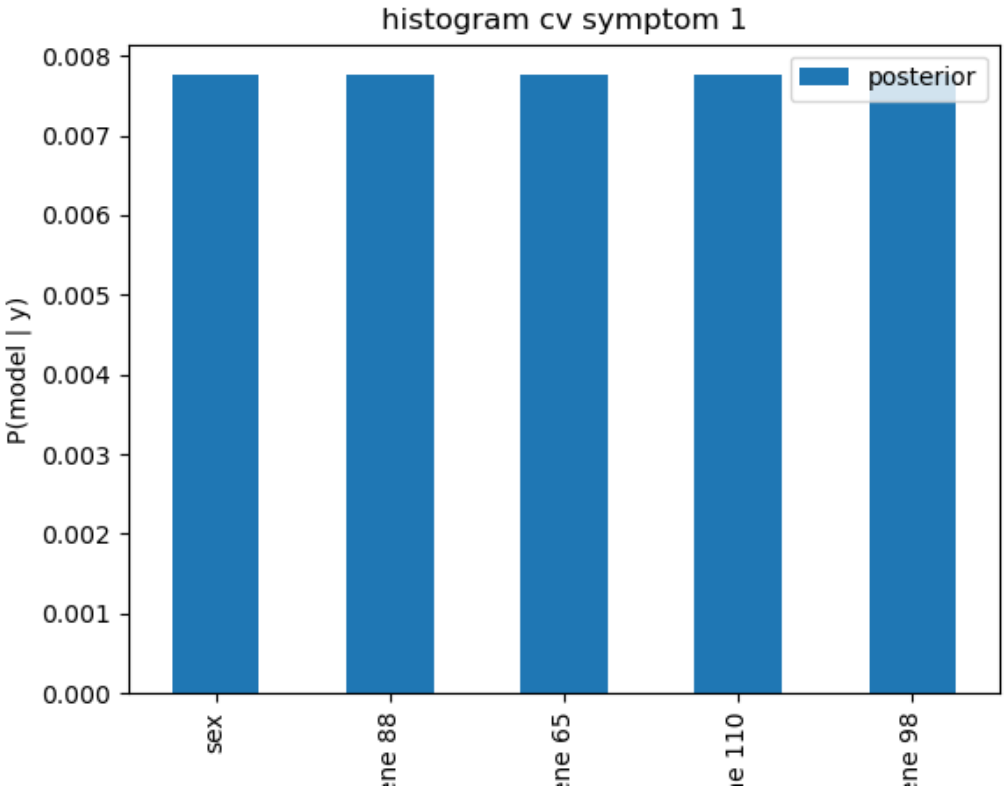
[17]:
```
display(Image(filename="img/histogram_for_symptom_2.png", width = 600))
```



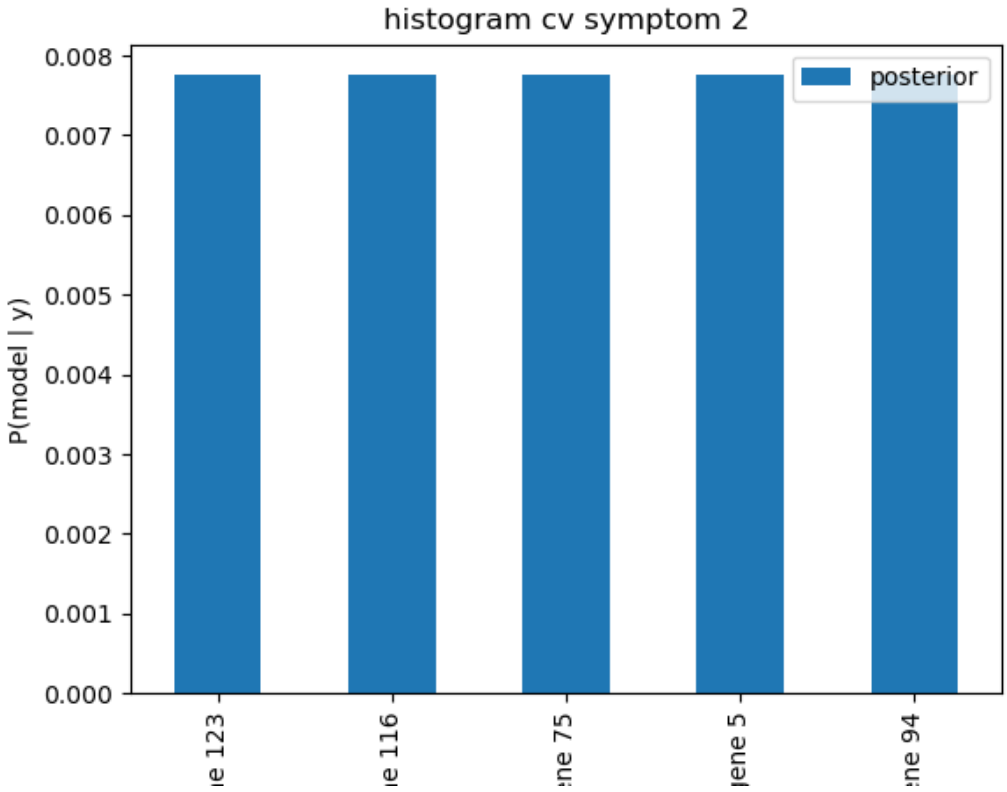
From the plots of the posterior probability for the model, we see that the full model has the highest posterior probability of the different models. This is likely related to the fact that the likelihood of the data is calculated using the model that was also fitted with this data. We see that the single covariates 'gene 4' and 'gene 3' are the models with the highest posterior probability for symptom 1 and symptom 2 respectively. This seems to support the accuracy score from the part above.

Experimental hybrid hierarchical We also use cross-validation in order to reduce the overfitting, this turns into a hybrid version which is using training data to fit a model and then calculates the likelihood of test data by using this model. The results are shown below.

[14]:
```
display(Image(filename="img/histogram_cv_symptom_1.png", width = 600))
```



[15]:
```
display(Image(filename="img/histogram_cv_symptom_2.png", width = 600))
```



Factors that may be important for disease epidemiology Based on the variable selection using stratified cross-validation and logistic regression above it could be important to look further into the gene expression that were selected using the variable selection above for the different symptoms. Since the predictive ability of the model seems to be best (on the validation set within the cross-validation) it implies that these genes should be further investigated. This could point toward there being a single-cause model and intuitively, if the data of the selected genes could be measured more accurately/effectively, this could be important for disease epidemiology.

In an ideal world, we would know if we found a sufficient covariate for the outcome y_t. This would imply that the outcome would be conditionally independent of the policy given this covariate and the action a_t (Dimitrakakis, 2020, pp. 137-138). Intuitively, if we knew that for example 'gene 3' was the sufficient covariate, we would need only to check this covariate for the different observations.

More likely is maybe the case where one of the covariates found above is an instrumental variable? In this case, the instrumental variable is a "representation" of the underlying/latent variable that affects both the outcome y_t and the action a_t (Dimitrakakis, 2020, p. 138). If this instrumental variable is possible to measure it could still function as a proxy for the sufficient variable (which may be unobservable).

1.2.2 Measuring the effect of actions

We now need to look into the action a and the outcome y followed by this action. Intuitively, what we would like to measure is $y_t|a_t$, that is, the outcome given the different actions (in this case therapeutic intervention). If we define $y_t = 1$ to be a positive outcome (positive outcome) and $y_t = 0$ a negative outcome (no effect of action). Then we could define $a_t = 0$ to be placebo (no intervention) and $a_t = 1$ to be the intervention/medication.

When looking at the outcome variable y_t we could think of this as a Bernoulli distributed variable when a_t is given. We could then look at the estimator for the utility of a policy given an action a_t. We then have

$$\hat{E}(U|a_t = a) = \frac{\sum_{t \in a_t = a} U(a_t, y_t)}{|a_t = a|}$$

adapted from (Dimitrakakis, 2020, pp. 140-141). For simplicity, we assume that we have an utility function $U(a_t = 1, y_t = 1) = 1$ which is, the observation were given a treatment ($a_t = 1$) and it did have a postive effect ($y_t = 1$). On the other hand $U(a_t = 1, y_t = 0) = 0$, the utility for treated observations which still has symptoms.

Similarily for those observations that got placebo treatment $a_t = 0$. This gives the following

$$\begin{aligned} U(a_t = 1, y_t = 1) &= 1 \\ U(a_t = 1, y_t = 0) &= 0 \\ U(a_t = 0, y_t = 1) &= 1 \\ U(a_t = 0, y_t = 0) &= 0 \end{aligned}$$

This approach will simply look at the action a and the outcome y from the action. So this measurement will only check $y_t|a_t$ independent of the historical data x_t. So in a sense it is only measuring the action a_t independently of the previous condition (symptoms) in x_t.

I then implemented the method

[1]:
```
def measure_effect(self, action):
        """Calculates the measured effect of an action.

        Args:
            action: 1 for treatment and 0 for placebo

        Returns:
            The measured effect.
        """
        y_joined = [self.y_train, self.y_test]
        y = pd.concat(y_joined)
        y_array = self._to_flat_array(y)

        a_joined = [self.a_train, self.a_test]
        a = pd.concat(a_joined)
```

```
        a_array = self._to_flat_array(a)

        return self._utility(a_array, y_array, action)
```

with the (private) helper method

[2]:
```
def _utility(self, a, y, at):
        """Calculates utility.

        Args:
            a: action array
            y: outcome array
            at: action to measure utility for

        Returns:
            Utility for observation.
        """
        num_at = len(np.where(a == at)[0])
        u = 0

        for i in range(len(a)):
            if a[i] == at and y[i] == 1:
                u += 1

        return u/num_at
```

running this on the entire dataset gave expected utility from the treatment $a_t = 1$ to be ≈ 66 times larger than the expected utility from the placebo $a_t = 0$.

From the fact that the expected utility for active treatment is higher than the expected utility from the placebo, it could indicate that the active treatment should be recommended. We try to separate the 'symptom 1' cases from the 'symptom 2' cases. We then want to examine the expected utility of active treatment ($a_t = 1$) versus placebo ($a_t = 0$) and condition on the symptom. Trying to find this out, we implemented the following method

[1]:
```
def measure_effect_symptom(self, action, symptom):
        """Calculates the measured effect of an action.

        Args:
            action: 1 for treatment and 0 for placebo
            symptom: separate observations based on symptom

        Returns:
            The measured effect.
        """
        y_joined = [self.y_train, self.y_test]
        y = pd.concat(y_joined)
        y_array = self._to_flat_array(y)
```

```
        a_joined = [self.a_train, self.a_test]
        a = pd.concat(a_joined)
        a_array = self._to_flat_array(a)

        x_joined = [self.x_train, self.x_test]
        x = pd.concat(x_joined)

        if symptom == 1:
            sym_idx = x.iloc[:, -2] == 1
        else:
            sym_idx = x.iloc[:, -1] == 1

        a_cond_sym = a_array[sym_idx]
        y_cond_sym = y_array[sym_idx]

        return self._utility(a_cond_sym, y_cond_sym, action)
```

and got the following results (for the entire dataset):

E(U|a_t = 0, sym = 1) = 0.01427469135802469

E(U|a_t = 1, sym = 1) = 0.5964479526393686

E(U|a_t = 0, sym = 2) = 0.008955223880597015

E(U|a_t = 1, sym = 2) = 0.5956112852664577

this could maybe indicate that the active treatment has an effect in all the observed cases because the expected utility is relatively higher for the active treatment than placebo?

2 References

Borgan, Ø. (2019). Regresjon og konfundering - notat til STK1110. Retrieved from: https://www.uio.no/studier/emner/matnat/math/STK1110/h19/confounding-regresjon.pdf

Dawid, A. P. (2015). Statistical Causality from a Decision-Theoretic Perspective. Annual Review of Statistics and Its Application, 2(1), 273-303.

Dimitrakakis, C. (2020). Machine learning in science and society. Unpublished. Department of Informatics, University of Oslo.

Hastie, T., Tibshirani, R. & Friedman, J. (2016). The Elements of Statistical Learning. Data Mining, Inference and Prediciton. New York, NY: Springer New York.

James, G., Witten, D., Hastie, T. & Tibshirani, R. (2013). An Introduction to Statistical Learning (Vol. 103 Springer Texts in Statistics). New York, NY: Springer New York.

scikit learn. (2020, n.d.). sklearn.feature_selection.RFECV. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html#sklearn.feature_selection.RFECV

3 Appendices

3.1 Part 1

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, StratifiedKFold, KFold
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss


class MedicalData:
    def __init__(self):
        self._get_data()

    def _get_data(self):
        """Gets the data and places it into three dataframes stored as instance
        variables of the object.
        """
        x = pd.read_csv("./data/medical/historical_X.dat",
                        sep=" ", header=None)
        personal_columns = ["sex", "smoker"]
        gene_columns = ["gene " + str(i) for i in range(1, 127)]
        symptom_columns = ["symptom 1", "symptom 2"]
        x_columns = personal_columns + gene_columns + symptom_columns
        x.columns = x_columns

        y = pd.read_csv("./data/medical/historical_Y.dat",
                        sep=" ", header=None)
        y.columns = ["outcome"]

        a = pd.read_csv("./data/medical/historical_A.dat",
                        sep=" ", header=None)
        a.columns = ["action"]

        self.x_train, self.x_test, self.y_train, self.y_test, self.a_train,␣
↪self.a_test = train_test_split(
            x, y, a, test_size=0.3, random_state=1)

    def data_analysis(self):
        """

        """
        self.frequency_symptoms()
        self.variable_selection(3)
```

```
    def _plot_variable_selection(self, accuracy_score1, accuracy_score2,␣
↪show=False):
        """Plots the accuracy score for the different symptoms.

        Args:
            accuracy_score1: the accuracy score from the CV of symptom 1
            accuracy_score2: the accuracy score from the CV of symptom 2
            show: whether or not to show the plot
        """
        fig, (axis1, axis2) = plt.subplots(1, 2)
        fig.suptitle("Accuracy score for variable selection")

        axis1.plot(range(1, len(accuracy_score1) + 1), accuracy_score1)
        axis1.set_title("Symptom 1")
        axis1.set_ylabel("Accuracy score")
        axis1.set_xlabel("Number of covariates")
        axis2.plot(range(1, len(accuracy_score2) + 1), accuracy_score2)
        axis2.set_title("Symptom 2")
        axis2.set_xlabel("Number of covariates")

        if show:
            plt.show()
        else:
            plt.savefig("img/var_sel.png")

    def variable_selection(self, num_folds):
        """Performs variable selection using a num_folds cross-validation.

        RFECV adapted from https://scikit-learn.org/stable/auto_examples/
        feature_selection/plot_rfe_with_cross_validation.html#sphx-glr-auto-
        examples-feature-selection-plot-rfe-with-cross-validation-py

        Args:
            num_folds: the number of folds to use in the cross-validation
        """
        logistic_regression = LogisticRegression(max_iter=1000)
        variable_selection_cv = RFECV(
            estimator=logistic_regression, step=1,␣
↪cv=StratifiedKFold(num_folds, random_state=1), scoring='accuracy')

        x = self.x_train.iloc[:, : -2]
        symptom1 = self.x_train.iloc[:, -2]
        symptom2 = self.x_train.iloc[:, -1]

        # symptom 1
        variable_selection_cv.fit(x, symptom1)
```

```
        accuracy_symptom1 = variable_selection_cv.grid_scores_
        symptom1_indices = np.where(
            variable_selection_cv.support_ == True)[0]
        print(f"Symptom 1 covariates = {x.columns[symptom1_indices]}")

        # symptom 2
        variable_selection_cv.fit(x, symptom2)
        accuracy_symptom2 = variable_selection_cv.grid_scores_
        symptom2_indices = np.where(
            variable_selection_cv.support_ == True)[0]
        print(f"Symptom 2 covariates = {x.columns[symptom2_indices]}")

        self._plot_variable_selection(accuracy_symptom1, accuracy_symptom2)

    def frequency_symptoms(self, show=False):
        """Generates simple histogram showing the frequency of the different
        symptoms. Looks at the entire dataset when considering the frequency.

        Args:
            show: whether or not to show the plot
        """
        x_joined = [self.x_train, self.x_test]
        x = pd.concat(x_joined)

        fig, (axis1, axis2) = plt.subplots(1, 2)
        fig.suptitle("Histogram of symptoms")
        axis1.hist(x["symptom 1"], color='b')
        axis1.set_title("Symptom 1")
        axis2.hist(x["symptom 2"], color='r')
        axis2.set_title("Symptom 2")

        if show:
            plt.show()
        else:
            plt.savefig("img/freq_hist.png")

    def measure_effect(self, action):
        """Calculates the measured effect of an action.

        Args:
            action: 1 for treatment and 0 for placebo

        Returns:
            The measured effect.
        """
        y_joined = [self.y_train, self.y_test]
        y = pd.concat(y_joined)
```

```
        y_array = self._to_flat_array(y)

        a_joined = [self.a_train, self.a_test]
        a = pd.concat(a_joined)
        a_array = self._to_flat_array(a)

        return self._utility(a_array, y_array, action)

    def measure_effect_symptom(self, action, symptom):
        """Calculates the measured effect of an action.

        Args:
            action: 1 for treatment and 0 for placebo
            symptom: separate observations based on symptom

        Returns:
            The measured effect.
        """
        y_joined = [self.y_train, self.y_test]
        y = pd.concat(y_joined)
        y_array = self._to_flat_array(y)

        a_joined = [self.a_train, self.a_test]
        a = pd.concat(a_joined)
        a_array = self._to_flat_array(a)

        x_joined = [self.x_train, self.x_test]
        x = pd.concat(x_joined)

        if symptom == 1:
            sym_idx = x.iloc[:, -2] == 1
        else:
            sym_idx = x.iloc[:, -1] == 1

        a_cond_sym = a_array[sym_idx]
        y_cond_sym = y_array[sym_idx]

        return self._utility(a_cond_sym, y_cond_sym, action)

    def _to_flat_array(self, df):

        numpy_array = df.to_numpy()
        return numpy_array.flatten()

    def _utility(self, a, y, at):
        """Calculates utility.
```

```
        Args:
            a: action array
            y: outcome array
            at: action to measure utility for

        Returns:
            Utility for observation.
        """
        num_at = len(np.where(a == at)[0])
        u = 0

        for i in range(len(a)):
            if a[i] == at and y[i] == 1:
                u += 1

        return u/num_at

    def hierarchical_model(self, data, symptom):
        """Calculates the hierarchical model for the medical data.

        Args:
            data: the data to calculate the posterior probability
            symptom: which symptom to use as response variable

        Returns:
            Posterior probabilites in a Pandas dataframe.
        """
        x = data.iloc[:, : -2]
        if symptom == 1:
            symptom = data.iloc[:, -2]
        else:
            symptom = data.iloc[:, -1]

        num_models = len(x.iloc[0])
        log_likelihoods = np.zeros(num_models + 1)
        model = LogisticRegression(max_iter=500)

        for i in range(0, num_models + 1):
            if i != num_models:
                single_column = x.iloc[:, i].to_numpy()
                single_covariate = single_column.reshape(-1, 1)
                log_reg = model.fit(single_covariate, symptom)
                p_t = log_reg.predict_proba(single_covariate)
                log_likelihoods[i] = -log_loss(symptom, p_t)
            else:
                log_reg = model.fit(x, symptom)
                p_t = log_reg.predict_proba(x)
```

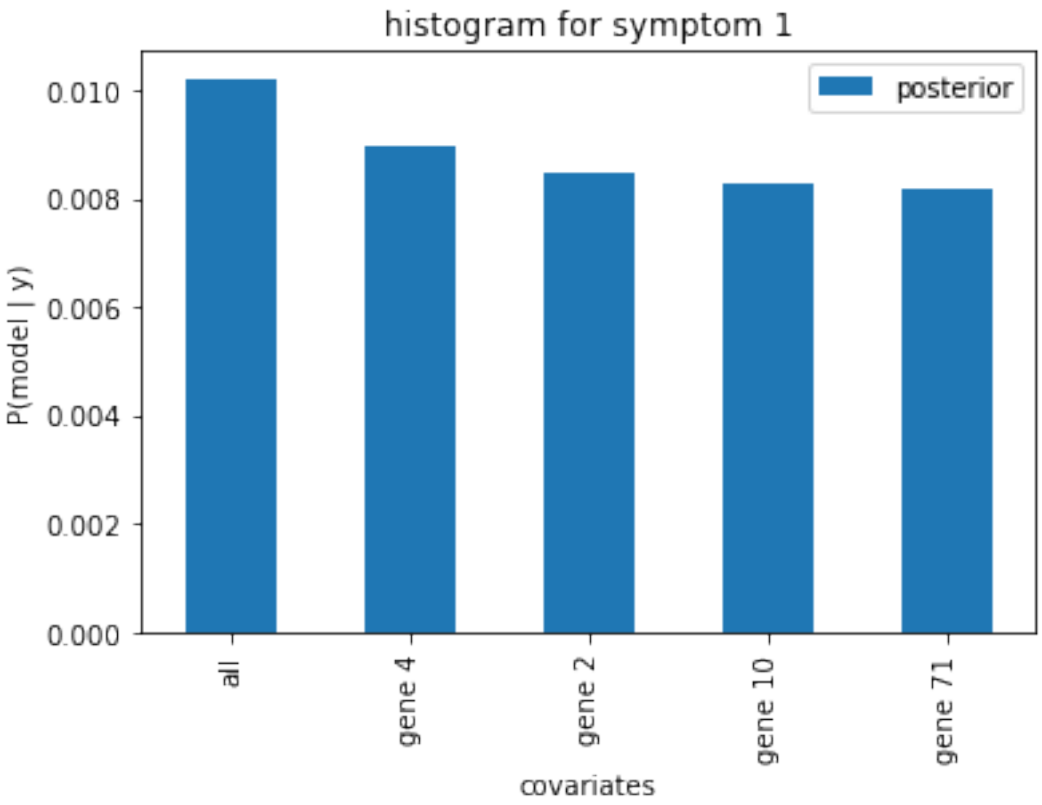
```
                log_likelihoods[i] = -log_loss(symptom, p_t)

        # calculating the posterior
        likelihood = np.exp(log_likelihoods)
        prior = np.repeat(1/(num_models + 1), num_models + 1)
        p_y = np.sum(likelihood*prior)
        posterior = (likelihood*prior)/p_y

        # constructing the dataframe
        last_index = pd.Index(data=["all"])
        model_names = x.columns.append(last_index)
        posterior_df = pd.DataFrame(data=posterior, index=model_names)
        posterior_df.columns = ["posterior"]

        return posterior_df

    def _calculate_posterior(self, xtrain, xtest, ytrain, ytest):
        """Calculates the posterior of a test set using a model fitted on
        training data.

        Args:
            xtrain: training covariates
            xtest: test covariates
            ytrain: training response
            ytest: test response

        Returns:
            The posterior probability of the different models.
        """
        num_models = len(xtrain.iloc[0])
        log_likelihoods = np.zeros(num_models + 1)

        model = LogisticRegression(max_iter=500)

        for i in range(0, num_models + 1):
            if i != num_models:
                single_column = xtrain.iloc[:, i].to_numpy()
                single_covariate = single_column.reshape(-1, 1)
                log_reg = model.fit(single_covariate, ytrain)

                single_column_test = xtest.iloc[:, i].to_numpy()
                single_covariate_test = single_column_test.reshape(-1, 1)
                p_t = log_reg.predict_proba(single_covariate_test)

                log_likelihoods[i] = -log_loss(ytest, p_t)
            else:
                log_reg = model.fit(xtrain, ytrain)
```

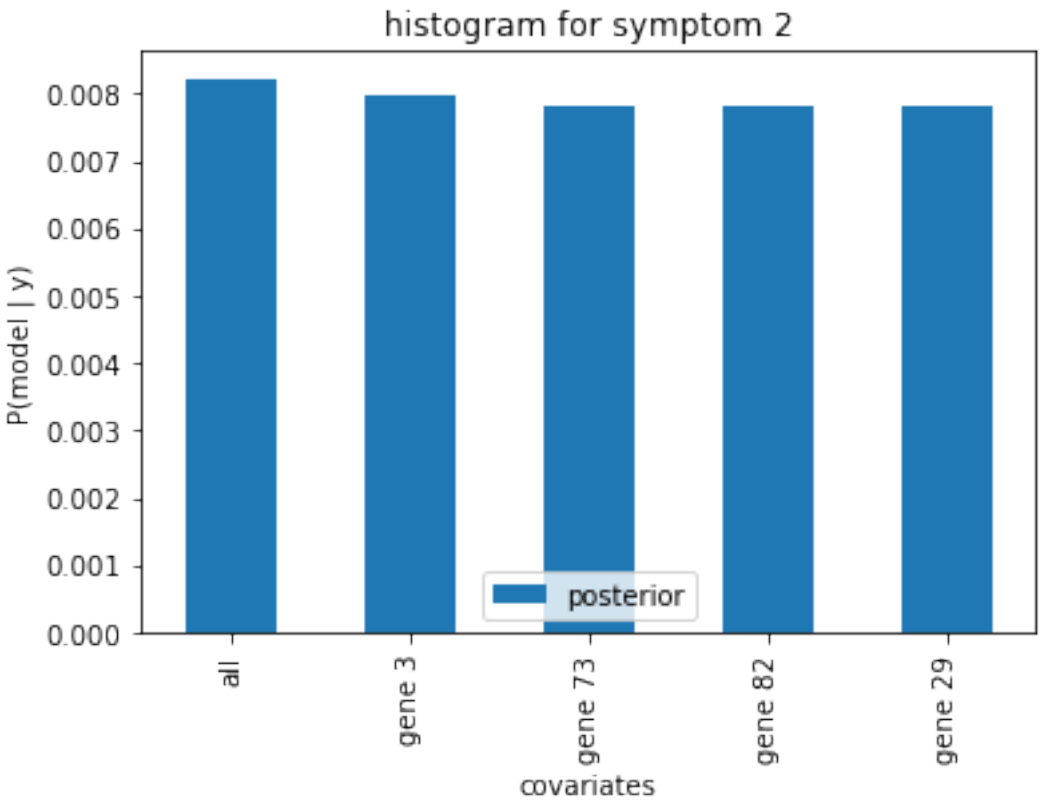
```
            p_t = log_reg.predict_proba(xtest)
            log_likelihoods[i] = -log_loss(ytest, p_t)

    likelihood = np.exp(log_likelihoods)
    prior = np.repeat(1/(num_models + 1), num_models + 1)
    p_y = np.sum(likelihood*prior)
    posterior = (likelihood*prior)/p_y

    return posterior

def hierarchical_model_cv(self, symptom, k):
    """Calculates the hierarchical model for the medical data.

    Args:
        symptom: which symptom to use as response variable
        k: the number of folds to use in the cross-validation

    Returns:
        Posterior probabilites in a Pandas dataframe.
    """

    num_models = len(self.x_train.iloc[0]) - 1
    # (folds, models)
    cv_posterior = np.zeros((k, num_models))

    x_joined = [self.x_train, self.x_test]
    x_raw = pd.concat(x_joined)
    x = x_raw.iloc[:, : -2]

    if symptom == 1:
        y = x.iloc[:, -2]
    else:
        y = x.iloc[:, -1]

    kf = KFold(n_splits=k, shuffle=True)
    k_counter = 0

    for train_indices, test_indices in kf.split(x):
        xtrain = x.iloc[train_indices, :]
        ytrain = y[train_indices]
        xtest = x.iloc[test_indices, :]
        ytest = y[test_indices]

        posterior = self._calculate_posterior(xtrain, xtest, ytrain, ytest)
        cv_posterior[k_counter, :] = posterior
        k_counter += 1
```

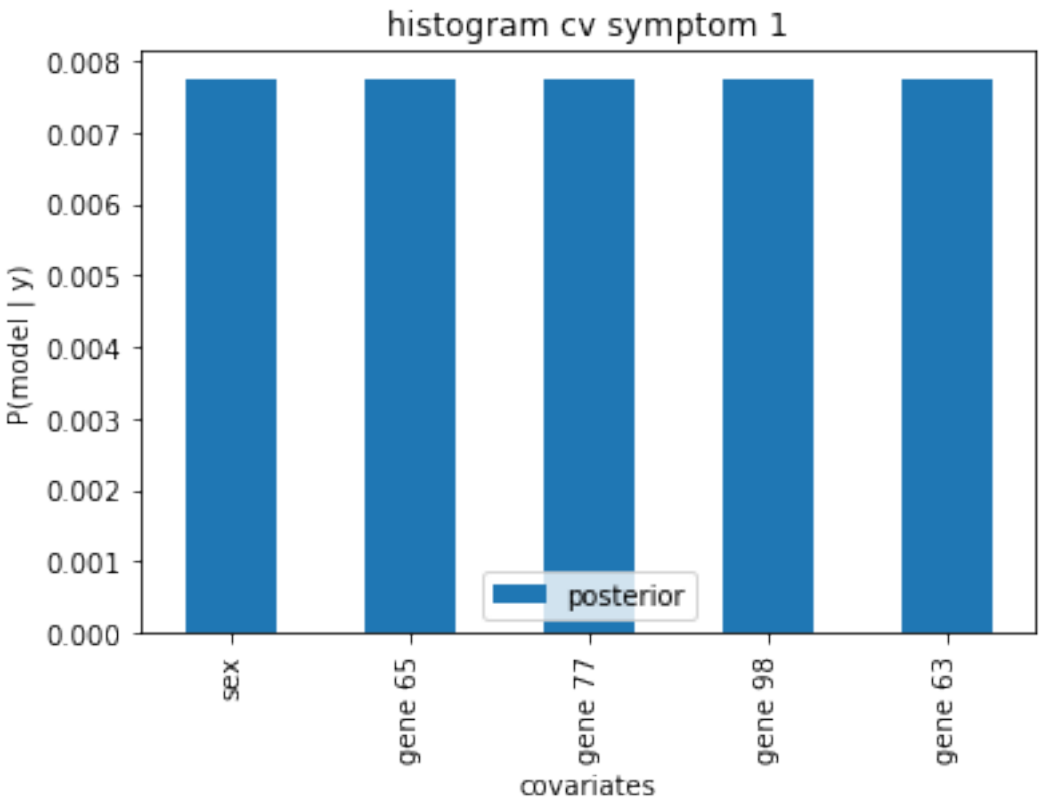
```
        posterior = np.mean(cv_posterior, 0)

        # constructing the dataframe
        last_index = pd.Index(data=["all"])
        model_names = x.columns.append(last_index)
        posterior_df = pd.DataFrame(data=posterior, index=model_names)
        posterior_df.columns = ["posterior"]

        return posterior_df


def plot_posteriors(posteriors, num, title, show=True):
    """Plots the top k posterios.

    Args:
        posteriors: sorted Pandas dataframe with posteriors
        num: the number of posteriors to plot
        title: title of the histogram
        show: whether or not to show the plot
    """
    plot_posteriors = posteriors.sort_values(
        by="posterior", ascending=False)[:num]

    plot_posteriors.plot.bar()
    plt.title(title)
    plt.xlabel("covariates")
    plt.ylabel("P(model | y)")

    if show:
        plt.show()
    else:
        filename = title.replace(" ", "_") + ".png"
        plt.savefig("img/" + filename)


if __name__ == "__main__":
    data = MedicalData()
    # data.data_analysis()
    expected_utility_1 = data.measure_effect(1)
    expected_utility_0 = data.measure_effect(0)
    print(f"E[U|a_t = 1] = {expected_utility_1}")
    print(f"E[U|a_t = 0] = {expected_utility_0}")

    util_sym1_a1 = data.measure_effect_symptom(1, 1)
    util_sym2_a1 = data.measure_effect_symptom(1, 2)
    util_sym1_a0 = data.measure_effect_symptom(0, 1)
```

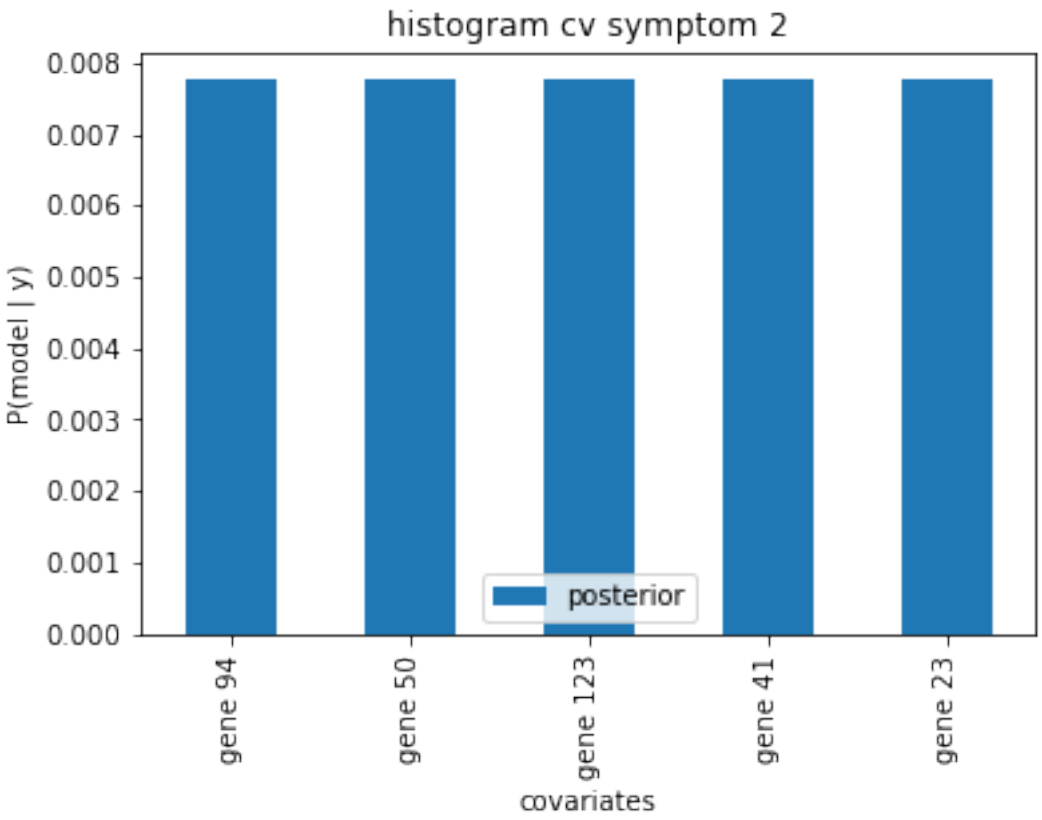
```
    util_sym2_a0 = data.measure_effect_symptom(0, 2)
    print(f"E[U|a_t = 1, sym = 1] = {util_sym1_a1}")
    print(f"E[U|a_t = 1, sym = 2] = {util_sym2_a1}")
    print(f"E[U|a_t = 0, sym = 1] = {util_sym1_a0}")
    print(f"E[U|a_t = 0, sym = 2] = {util_sym2_a0}")

    x_joined = [data.x_train, data.x_test]
    x = pd.concat(x_joined)
    sym1_posteriors = data.hierarchical_model(x, 1)
    plot_posteriors(sym1_posteriors, 5, "histogram for symptom 1", show=False)

    sym2_posteriors = data.hierarchical_model(x, 2)
    plot_posteriors(sym2_posteriors, 5, "histogram for symptom 2", show=False)

    sym1_cv_posteriors = data.hierarchical_model_cv(1, 5)
    plot_posteriors(sym1_cv_posteriors, 5,
                    "histogram cv symptom 1", show=False)

    sym2_cv_posteriors = data.hierarchical_model_cv(2, 5)
    plot_posteriors(sym2_cv_posteriors, 5,
                    "histogram cv symptom 2", show=False)
```

```
E[U|a_t = 1] = 0.5883376849434291
E[U|a_t = 0] = 0.008958712022851208
E[U|a_t = 1, sym = 1] = 0.5964479526393686
E[U|a_t = 1, sym = 2] = 0.5956112852664577
E[U|a_t = 0, sym = 1] = 0.01427469135802469
E[U|a_t = 0, sym = 2] = 0.008955223880597015
```


histogram for symptom 1
posterior
0.010
0.008
0.006
0.004
0.002
0.000
P(model | y)
all
gene 4
gene 2
gene 10
gene 71
covariates


histogram for symptom 2
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
P(model | y)
posterior
all
gene 3
gene 73
gene 82
gene 29
covariates


histogram cv symptom 1
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
P(model | y)
posterior
sex
gene 65
gene 77
gene 98
gene 63
covariates


histogram cv symptom 2
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
P(model | y)
posterior
gene 94
gene 50
gene 123
gene 41
gene 23
covariates

[ ]: