Question 1:

A)

$S^{(i)}$ represents our train sample S but with i'th observation replaced by different one, (x', y')

$A(S)$ is the w that minimizes regularized loss, $L_s(w) + \lambda\|w\|^2$

B)

We can take our the i'th observation out of sum in L_S and switch it with (x', y'):

$$\begin{aligned}
f_S(v) - f_S(u) &= L_S(v) + \lambda\|v\|^2 - L_S(u) - \lambda\|u\|^2 \\
&= \frac{1}{m}\sum_{j=1}^{m} l(v, x_j, y_j) - \frac{1}{m}\sum_{j=1}^{m} l(u, x_j, y_j) + \lambda\|v\|^2 - \lambda\|u\|^2 \\
&= \frac{1}{m}\sum_{j\neq i}^{m} l(v, x_j, y_j) + \frac{l(v, x', y')}{m} - \frac{l(v, x', y')}{m} + \frac{l(v, x_i, y_i)}{m} - \frac{1}{m}\sum_{j\neq i}^{m} l(u, x_j, y_j) \\
&- \frac{l(u, x', y')}{m} + \frac{l(u, x', y')}{m} - \frac{l(u, x_i, y_i)}{m} + \lambda\|v\|^2 - \lambda\|u\|^2 \\
&= L_{S^{(i)}}(v) - \frac{l(v, x', y')}{m} + \frac{l(v, x_i, y_i)}{m} - L_{S^{(i)}}(u) + \frac{l(u, x', y')}{m} - \frac{l(u, x_i, y_i)}{m} \\
&+ \lambda\|v\|^2 - \lambda\|u\|^2 \\
&= L_{S^{(i)}}(v) + \lambda\|v\|^2 - \left(L_{S^{(i)}}(u) + \lambda\|u\|^2\right) + \frac{l(v, x_i, y_i) - l(u, x_i, y_i)}{m} \\
&+ \frac{l(u, x', y') - l(v, x', y')}{m}
\end{aligned}$$

C)

$$\begin{aligned}
f_S\left(A\left(S^{(i)}\right)\right) - f_S(A(S)) \\
&= L_{S^{(i)}}\left(A\left(S^{(i)}\right)\right) + \lambda\left\|A\left(S^{(i)}\right)\right\|^2 - \left(L_{S^{(i)}}(A(S)) + \lambda\|A(S)\|^2\right) \\
&+ \frac{l\left(A\left(S^{(i)}\right), x_i, y_i\right) - l(A(S), x_i, y_i)}{m} + \frac{l(A(S), x', y') - l\left(A\left(S^{(i)}\right), x', y'\right)}{m}
\end{aligned}$$

Since $A\left(S^{(i)}\right)$ is the one that minimizes $f_{S^{(i)}}$,

$$L_{S^{(i)}}\left(A\left(S^{(i)}\right)\right) + \lambda\left\|A\left(S^{(i)}\right)\right\|^2 \leq L_{S^{(i)}}(A(S)) + \lambda\|A(S)\|^2$$

$$L_{S^{(i)}}\left(A\left(S^{(i)}\right)\right) + \lambda\left\|A\left(S^{(i)}\right)\right\|^2 - \left(L_{S^{(i)}}(A(S)) + \lambda\|A(S)\|^2\right) \leq 0$$

$$\begin{aligned}
\Rightarrow L_{S^{(i)}}\left(A\left(S^{(i)}\right)\right) + \lambda\left\|A\left(S^{(i)}\right)\right\|^2 - \left(L_{S^{(i)}}(A(S)) + \lambda\|A(S)\|^2\right) \\
&+ \frac{l\left(A\left(S^{(i)}\right), x_i, y_i\right) - l(A(S), x_i, y_i)}{m} + \frac{l(A(S), x', y') - l\left(A\left(S^{(i)}\right), x', y'\right)}{m} \\
&\leq \frac{l\left(A\left(S^{(i)}\right), x_i, y_i\right) - l(A(S), x_i, y_i)}{m} + \frac{l(A(S), x', y') - l\left(A\left(S^{(i)}\right), x', y'\right)}{m}
\end{aligned}$$

D)

As we have seen in lectures, $\lambda\|w\|^2$is 2λ strongly convex, and thus f_S is 2λ strongly convex as sum of $\lambda\|w\|^2$ and L_S.

From that, as we have seen in lecture, since $A(S)$ is minimizer of f_S,

$$f_S\left(A\left(S^{(i)}\right)\right) - f_S(A(S)) \geq \frac{2\lambda}{2}\left\|A\left(S^{(i)}\right) - A(S)\right\|^2 = \lambda\left\|A\left(S^{(i)}\right) - A(S)\right\|^2$$

And from C), we get:

$$\lambda\left\|A\left(S^{(i)}\right) - A(S)\right\|^2 \leq f_S\left(A\left(S^{(i)}\right)\right) - f_S(A(S))$$
$$\leq \frac{l\left(A\left(S^{(i)}\right), x_i, y_i\right) - l(A(S), x_i, y_i)}{m} + \frac{l(A(S), x', y') - l\left(A\left(S^{(i)}\right), x', y'\right)}{m}$$

E)

$$\lambda\left\|A\left(S^{(i)}\right) - A(S)\right\|^2 \leq \frac{l\left(A\left(S^{(i)}\right), x_i, y_i\right) - l(A(S), x_i, y_i)}{m} + \frac{l(A(S), x', y') - l\left(A\left(S^{(i)}\right), x', y'\right)}{m}$$
$$\leq \frac{\rho \cdot \left\|A\left(S^{(i)}\right) - A(S)\right\|}{m} + \frac{\rho \cdot \left\|A\left(S^{(i)}\right) - A(S)\right\|}{m} = \frac{2\rho\left\|A\left(S^{(i)}\right) - A(S)\right\|}{m}$$

$$\Rightarrow \left\|A\left(S^{(i)}\right) - A(S)\right\| \leq \frac{2\rho}{\lambda m}$$

F)

Using result from E),

$$l\left(A\left(S^{(i)}\right), x_i, y_i\right) - l(A(S), x_i, y_i) = \rho\left\|A\left(S^{(i)}\right) - A(S)\right\| \leq \frac{2\rho^2}{\lambda m}$$

G)

$L_D(w)$ is the expected loss (expected risk) of w given that our data comes from distribution D

$L_S(w)$ is the average loss on the sample that we are given (empirical risk).

We can calculate L_S given S and w, it is just mean on losses. However, we cannot calculate L_D since we don't know from which distribution our data comes from.

H)

As we have seen in lecture,

$$E_{S\sim D^m}\left[L_D(A(S))\_L_S\left(A(S)\right)\right] = E_{(S,(x',y'))\sim D^{m+1}, i\sim U(m)}\left[l\left(A\left(S^{(i)}\right), (x_i, y_i)\right) - l\left(A(S), (x_i, y_i)\right)\right]$$
$$\leq E\left[\frac{2\rho^2}{\lambda m}\right] = \frac{2\rho^2}{\lambda m}$$

Where the last inequality holds from E).

Q2

A)

$$\textbf{Precision} = \frac{TP}{TP + FP}$$

Precision is the ratio of true positives to the total true and false positives. This means that precision measures how many of the items classified as positive are actually positive.

$$\textbf{TPR = Recall} = \frac{TP}{TP + FN}$$

TPR = Recall is the ratio of true positives to the true positives and false negatives. In other words, we penalize recall for every mistake made on negative classifications.

Given these two metrics, we would like to maximize both of them.

B) **Recall** is more important when we can't afford to make mistakes relative to making True Negatives. For example: Email Spam, it's better to by mistake miss classify legitimate emails as spam (TN) rather than letting spam emails go through and be sent to the inbox meaning we want to minimize (FN).

Precision is more important when we don't necessarily mind making a few mistakes and would rather maximize correct choices and minimize the mistakes that we make. For example, obstacle detection- it would be better to prioritize FP as it's better to miss classify an object than not classify a real object since the impact of a autonomous car hitting an obstacle can be significant.

D)

Let's denote $\sigma(x) := \frac{e^x}{1+e^x}, w = (-0.3, -0.5, 0.5)$

$P_w(y = 1|x) = \mathrm{p}(\sigma(\langle w, x\rangle)) = \mathrm{P}(-0.3 - 0.5x_1 + 0.5x_2)$

Using a calculator we can calculate for each example:

$P_w(y = 1|x_0) = 0.03557$
$P_w(y = 1|x_1) = 0.66818$
$P_w(y = 1|x_2) = 0.00067$
$P_w(y = 1|x_3) = 0.214$
$P_w(y = 1|x_4) = 0.0218$

	intervals	FPR	TPR
0	0.000675	1.0	1.0
1	0.021881	1.0	0.5
2	0.668188	0.0	0.5
3	1.000000	0.0	0.0

Table 1: Table showing with intervals, FPR, and TPR values

E)

For the probabilities that we calculated in the previous part shown in the table above. Given the points and probabilities we"ll take the intervals and it's easy to see the values for the FPR and TPR accordingly

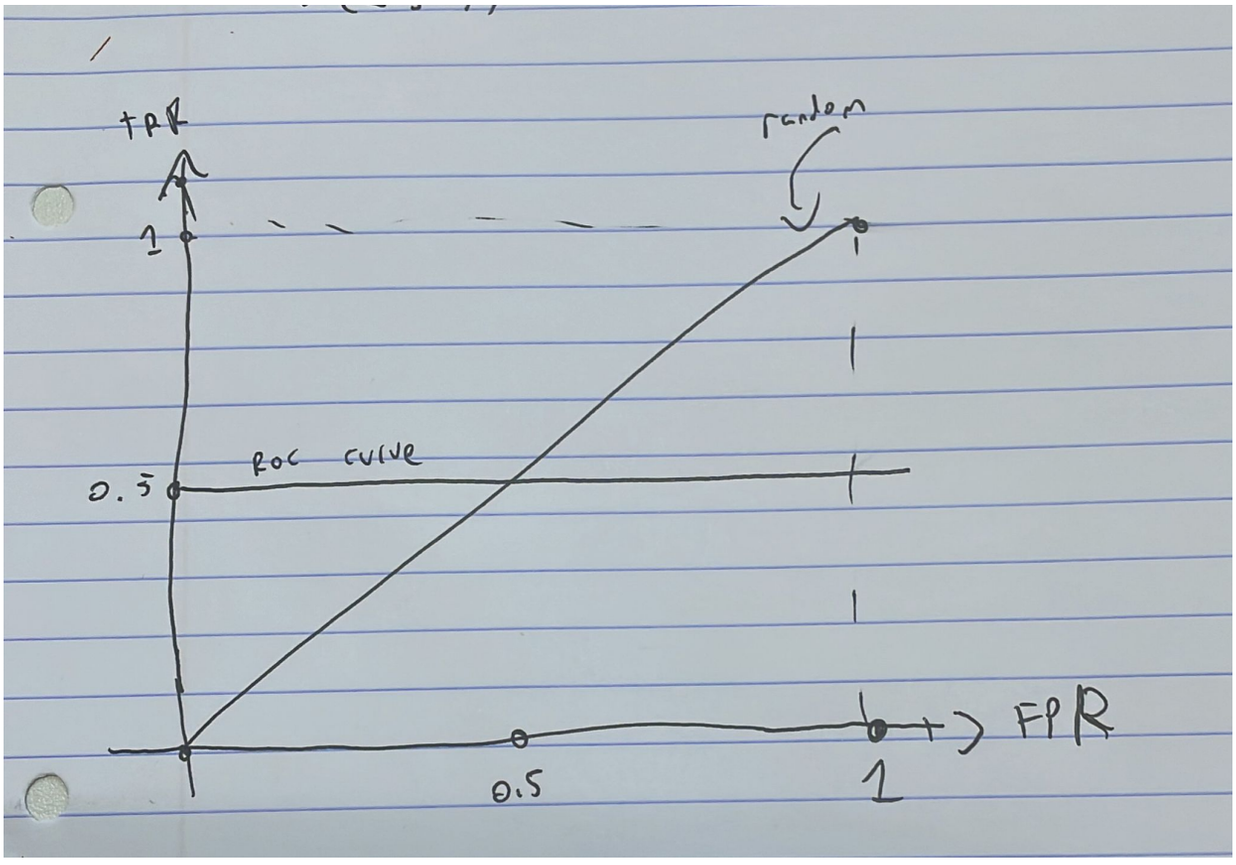


Figure 1: ROC for given example with random graph

F) The AUC-ROC is 0.5 as it's beneath the graph which is the same area that we would get by choosing randomly thus the model is **not** better than the random model.

Question 3

In [14]:

```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
```

In [15]:

```
def fetch_mnist():
    # Download MNIST dataset
    X, y = fetch_openml('Fashion-MNIST', version=1, return_X_y=True)
    X = X.to_numpy()
    y = y.to_numpy()

    # Randomly sample 7000 images
    np.random.seed(2)
    indices = np.random.choice(len(X), 7000, replace=False)
    X, y = X[indices], y[indices]
    return X, y
```

In [16]:

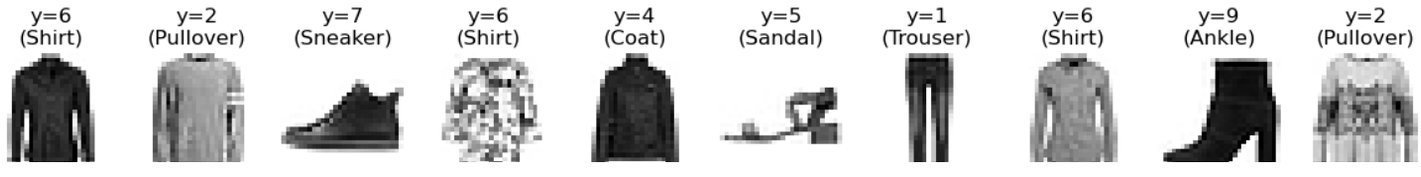
```
X, y = fetch_mnist()
print(X.shape, y.shape)
```

```
C:\Users\yumif\anaconda3\envs\mlcourse\lib\site-packages\sklearn\datasets\_openml.py:968: FutureWarning: The default value of `parser` will change from `'liac-arff'` to `'auto'` in 1.4. You can set `parser='auto'` to silence this warning. Therefore, an `ImportError` will be raised from 1.4 if the dataset is dense and pandas is not installed. Note that the pandas parser may return different data types. See the Notes Section in fetch_openml's API doc for details.
  warn(
```

```
(7000, 784) (7000,)
```

In [17]:

```
idx2class = {
    '0': 'T-shirt/top',
    '1': 'Trouser',
    '2': 'Pullover',
    '3': 'Dress',
    '4': 'Coat',
    '5': 'Sandal',
    '6': 'Shirt',
    '7': 'Sneaker',
    '8': 'Bag',
    '9': 'Ankle'
}

fig, axes = plt.subplots(1, 10, figsize=(15, 5))
for i in range(10):
    axes[i].imshow(X[i].reshape(28, 28), cmap="binary")
    axes[i].axis('off')
    axes[i].set_title(f"y={y[i]}\n({idx2class[y[i]]})")

plt.show()
```



In [48]:

```
def cross_validation_error(X, y, model, folds):
    fold_size = X.shape[0] // folds
    sum_train_error = 0
    sum_val_error = 0

    # Shuffle data (no bias by the initial order)
    permutation = np.random.permutation(len(y))
    X = X[permutation]
    y = y[permutation]

    for i in range(folds):
        v_start = i * fold_size
        v_end = (i + 1) * fold_size

        X_val = X[v_start:v_end]
        y_val = y[v_start:v_end]

        X_train = np.concatenate((X[:v_start], X[v_end:]), axis=0)
        y_train = np.concatenate((y[:v_start], y[v_end:]), axis=0)

        model.fit(X_train, y_train)
        y_val_pred = model.predict(X_val)
        y_train_pred = model.predict(X_train)

        sum_train_error += 1 - np.mean(y_train == y_train_pred)
        sum_val_error += 1 - np.mean(y_val == y_val_pred)

    return sum_train_error / folds, sum_val_error / folds
```

In [55]:

```
def SVM_results(X_train, y_train, X_test, y_test):
    # Define the set of parameters to test
    parameter_grid = [
        {"kernel": "linear"},
        {"kernel": "poly", "degree": 2},
        {"kernel": "poly", "degree": 4},
        {"kernel": "poly", "degree": 6},
        {"kernel": "poly", "degree": 8},
        {"kernel": "rbf", "gamma": 0.001},
        {"kernel": "rbf", "gamma": 0.01},
        {"kernel": "rbf", "gamma": 0.1},
        {"kernel": "rbf", "gamma": 1.0},
        {"kernel": "rbf", "gamma": 10},
    ]

    num_folds = 4
    performance_results = {}

    for params in parameter_grid:
        # Initialize the SVM model with current parameters
        model = SVC()
        model.set_params(**params)

        # Perform cross-validation
        train_cv_error, validation_cv_error = cross_validation_error(X_train, y_train, m
odel, num_folds)

        # Test error calculation
        y_test_pred = model.predict(X_test)
        test_error_rate = 1 - np.mean(y_test == y_test_pred)

        # Store the results
        key_name = f"{params['kernel']} kernel, "
        if params.get('degree') is not None:
            key_name += f"d = {params['degree']}"
        elif params.get('gamma') is not None:
```

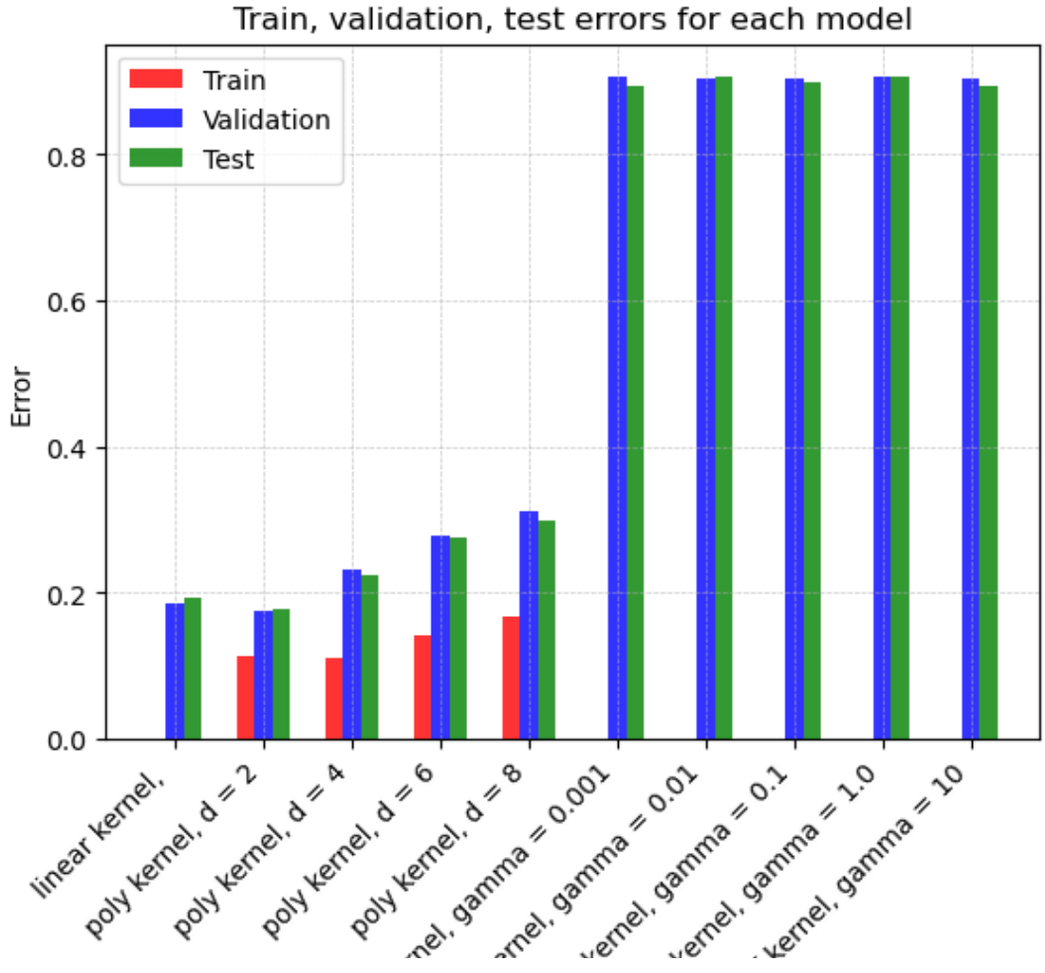
```
            key_name += f"gamma = {params['gamma']}"

        performance_results[key_name] = (train_cv_error, validation_cv_error, test_error
_rate)

    return performance_results
```

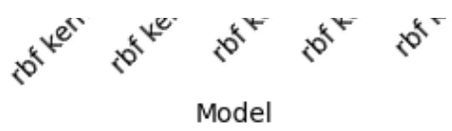
In [56]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=4
2)
results = SVM_results(X_train, y_train, X_test, y_test)
```

In [57]:

```
fig, ax = plt.subplots()

train_errors = [v[0] for v in results.values()]
val_error = [v[1] for v in results.values()]
test_errors = [v[2] for v in results.values()]

bar_width = 0.2
index = np.arange(len(results.keys()))
ax.grid(True, which='both', linestyle='--', linewidth=0.5, alpha=0.7)

bar_train = ax.bar(index - bar_width, train_errors, bar_width, label='Train', color='red
', alpha=0.8)
bar_val = ax.bar(index, val_error, bar_width, label='Validation', color='blue', alpha=0.
8)
bar_test = ax.bar(index + bar_width, test_errors, bar_width, label='Test', color='green'
, alpha=0.8)

ax.set_xlabel('Model')
ax.set_ylabel('Error')
ax.set_title('Train, validation, test errors for each model')
ax.set_xticks(index)
ax.set_xticklabels(results.keys(), rotation=45, ha='right')
ax.legend()

plt.show()
```





According to CV method, SVM with poly kernel and degree 2 is the best model, lowest (average) error on validation set. The best model on test set is the same model.

In [ ]: