WiDS Kalman Filtered Trend Trader Assignment 1

Aarav Malde and Krishang Krishna

December 2025

Question 1: Linear Regression

0.1 1. Multiple Linear Regression Model

The multiple linear regression model with p predictors is:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i$$

In matrix form:

$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$

Variables and Parameters:

- $\mathbf{y} = (y_1, \ldots, y_n)^\top$: The $n \times 1$ response variable vector.
- $\beta = (\beta_0, \beta_1, \ldots, \beta_p)^\top$: The vector of regression coefficients.
- $\mathbf{X}$: The $n \times (p+1)$ design matrix.
- ϵ: The $n \times 1$ vector of error terms.

Assumptions (Core Gauss-Markov):

1. Linearity in parameters.
2. Zero Conditional Mean: $\mathbb{E}[\epsilon|\mathbf{X}] = \mathbf{0}$.
3. Homoscedasticity and No Autocorrelation: $\mathrm{Var}(\epsilon|\mathbf{X}) = \sigma^2\mathbf{I}$.
4. No Perfect Multicollinearity: $\mathrm{rank}(\mathbf{X}) = p + 1$.

0.2 2. OLS Minimization and MSE Objective Function

Ordinary Least Squares (OLS) minimizes the **Sum of Squared Errors (SSE)**:

$$\mathrm{SSE}(\beta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

The full Mean-Squared Error (MSE) objective function $J(\beta)$ is:

$$J(\beta) = \frac{1}{n}(\mathbf{y} - \mathbf{X}\beta)^\top(\mathbf{y} - \mathbf{X}\beta)$$

0.3 3. Derivation of the OLS Estimator $\hat{\beta}$

We minimize $\mathrm{SSE}(\beta)$ by setting the gradient to zero:

$$\nabla_\beta \mathrm{SSE}(\beta) = -2\mathbf{X}^\top\mathbf{y} + 2\mathbf{X}^\top\mathbf{X}\beta = \mathbf{0}$$

Solving the Normal Equations:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

0.4 4. Conditions for Invertibility and Multicollinearity

$\mathbf{X}^\top\mathbf{X}$ is invertible if and only if $\mathbf{X}$ has **full column rank** (rank($\mathbf{X}$) $= p+1$). **Multicollinearity** causes $\mathbf{X}^\top\mathbf{X}$ to be singular due to linear dependence among predictors.

0.5 5. Orthogonal Projection

The predicted vector $\hat{\mathbf{y}} = \mathbf{X}\hat{\beta}$ is the **orthogonal projection** of $\mathbf{y}$ onto the column space of $\mathbf{X}$.

Proof that $\mathbf{X}^T(\mathbf{y} - \hat{\mathbf{y}}) = \mathbf{0}$:

$$\mathbf{X}^\top(\mathbf{y} - \hat{\mathbf{y}}) = \mathbf{X}^\top\mathbf{y} - \mathbf{X}^\top(\mathbf{X}\hat{\beta}) = \mathbf{X}^\top\mathbf{y} - \mathbf{X}^\top\mathbf{y} = \mathbf{0}$$

0.6 6. Gradient of $J(\beta)$ and Batch Gradient Descent

Given $J(\beta) = \frac{1}{2n}||\mathbf{X}\beta - \mathbf{y}||^2$.

$$\nabla_\beta J(\beta) = \frac{1}{n}\mathbf{X}^\top(\mathbf{X}\beta - \mathbf{y})$$

Batch Gradient Descent Update Rule:

$$\beta^{(t+1)} = \beta^{(t)} - \eta \cdot \frac{1}{n}\mathbf{X}^\top(\mathbf{X}\beta^{(t)} - \mathbf{y})$$

0.7 7. - 11. Computational Tasks

Required Python Code:

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from scipy import stats

try:
    df = pd.read_csv('linear_regression_dataset.csv')
except FileNotFoundError:
    print("Error: Dataset not found. Using dummy data for structure.")
    np.random.seed(42); n_samples=300; X_data=np.random.rand(n_samples, 12); y_data=X_data @ (np.ara

X_data = df.drop('y', axis=1).values
y = df['y'].values.reshape(-1, 1)
X = sm.add_constant(X_data, prepend=True)
n = X.shape[0]; p_plus_1 = X.shape[1]

try:
    XTX = X.T @ X
    XTX_inv = np.linalg.inv(XTX)
    beta_hat_numpy = XTX_inv @ (X.T @ y)
    print("Q1.7: NumPy Beta Hat (Intercept, X1...X12):", beta_hat_numpy.flatten())

    model_sklearn = LinearRegression().fit(X_data, df['y'])
    beta_hat_sklearn = np.insert(model_sklearn.coef_, 0, model_sklearn.intercept_)
    print("Q1.7: Sklearn Beta Hat:", beta_hat_sklearn)

    y_hat = X @ beta_hat_numpy
    residuals = y - y_hat

    plt.figure(figsize=(10, 6))
    plt.scatter(y_hat, residuals, alpha=0.6)
    plt.hlines(y=0, xmin=y_hat.min(), xmax=y_hat.max(), color='red', linestyle='--')
```

```
    plt.title('Q1.8: Residuals vs Fitted Values')
    plt.xlabel('Fitted Values ($\hat{y}$)')
    plt.ylabel('Residuals ($y - \hat{y}$)')
    plt.savefig('Q1_8_Residuals_Plot.png')
    plt.show()

    plt.figure(figsize=(8, 8))
    stats.probplot(residuals.flatten(), dist="norm", plot=plt)
    plt.title('Q1.9: Q-Q Plot of Residuals')
    plt.savefig('Q1_9_QQ_Plot.png')
    plt.show()

    H = X @ XTX_inv @ X.T
    leverage = np.diag(H)
    leverage_threshold = 2 * p_plus_1 / n

    s_squared = np.sum(residuals**2) / (n - p_plus_1)
    cooks_distance = (residuals**2 / (s_squared * p_plus_1)) * (leverage / (1 - leverage)**2)
    cooks_threshold = 4 / n

    print(f"\nQ1.11: High Leverage Indices (h_ii > {leverage_threshold:.4f}): {np.where(leverage > l
    print(f"Q1.11: Influential Indices (Cook's D > {cooks_threshold:.4f}): {np.where(cooks_distance

except np.linalg.LinAlgError:
    print("\nMulticollinearity detected: X.T @ X is singular and cannot be inverted.")
```

Q1.8 Comment on Homoscedasticity: Homoscedasticity holds if the residual spread is constant across all fitted values.

Q1.9 Comment on Normality: Normality holds if the points in the Q-Q plot closely follow the 45° reference line.

0.8 10. Violations and Model Assumptions

- **Heteroscedasticity:** Leads to incorrect standard errors and invalid statistical inference.
- **Non-Normality:** Invalidates statistical inference in small samples.

0.9 12. Bias-Variance Decomposition

The expected squared prediction error is:

$$\mathbb{E}[(y - \hat{f}(x))^2] = \text{Bias}^2 + \text{Var} + \sigma^2$$

0.10 13. Omitted Variable Bias (OVB)

a) Derive $\mathbb{E}[\alpha_1]$:

$$\mathbb{E}[\hat{\alpha}_1] \approx \beta_1 + \beta_2 \cdot \frac{\text{Cov}(x_1, x_2)}{\text{Var}(x_1)}$$

b) Show how omitting x_2 biases α_1: The bias is $\text{Bias}(\hat{\alpha}_1) \approx \beta_2 \cdot \frac{\text{Cov}(x_1,x_2)}{\text{Var}(x_1)}$. Bias occurs if $\beta_2 \neq 0$ and $\text{Cov}(x_1, x_2) \neq 0$. **c) State conditions under which the bias disappears:** Bias disappears if either $\beta_2 = 0$ or $\text{Cov}(x_1, x_2) = 0$.

0.11 14. Multicollinearity Simulation

a) Compute the condition number of $\mathbf{X}^T\mathbf{X}$:

```
import numpy as np
import statsmodels.api as sm
```

```
np.random.seed(42)
n_samples = 1000
x1 = np.random.normal(10, 5, n_samples)
z = np.random.normal(0, 1, n_samples)
x2_high_corr = x1 + 0.1 * z

X_data = np.vstack([x1, x2_high_corr]).T
X = sm.add_constant(X_data, prepend=True)

XTX = X.T @ X
eigenvalues = np.linalg.eigvals(XTX)
max_lambda = np.max(eigenvalues)
min_lambda = np.min(eigenvalues[eigenvalues > 1e-10])

condition_number = max_lambda / min_lambda
print(f"Q1.14a: Condition Number: {condition_number:.2f}")
```

b) Show how variance of β increases with correlation:

```
XTX_high_inv = np.linalg.inv(X.T @ X)
var_high_corr = np.diag(XTX_high_inv)

x2_low_corr = x1 + 5 * z
X_low = sm.add_constant(np.vstack([x1, x2_low_corr]).T, prepend=True)
XTX_low_inv = np.linalg.inv(X_low.T @ X_low)
var_low_corr = np.diag(XTX_low_inv)

print(f"Q1.14b: Var(Beta) High Corr (Beta1, Beta2): ({var_high_corr[1]:.4f}, {var_high_corr[2]:.4f})
print(f"Q1.14b: Var(Beta) Low Corr (Beta1, Beta2): ({var_low_corr[1]:.4f}, {var_low_corr[2]:.4f})")
```

c) Explain why multicollinearity causes instability but not bias: Multicollinearity causes **instability** (high variance) because the inverse of the near-singular $\mathbf{X}^\top\mathbf{X}$ matrix has large elements. It does **not cause bias** as the OLS estimator remains unbiased under this condition.

Question 2: Salary Prediction & Bias Detection

0.12 1. - 4. Data Preparation and Splitting

Q2.4 Stratification: Stratify by `education_level`.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

try:
    df = pd.read_csv('salary_dataset.csv')
except FileNotFoundError:
    print("Error: Dataset not found. Cannot proceed.")
    exit()

df['age'].fillna(df['age'].median(), inplace=True)
df.dropna(inplace=True)

categorical_features = ['gender', 'education_level', 'job_title', 'industry', 'city', 'remote_worker
numeric_features = ['age', 'years_experience', 'performance_score', 'previous_companies']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', 'passthrough', numeric_features),
        ('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False), categorical_features)
    ]
)

X_data = df.drop('salary', axis=1)
y_data = df['salary']

X_train, X_test, y_train, y_test = train_test_split(
    X_data, y_data,
    test_size=0.2,
    random_state=42,
    stratify=X_data['education_level']
)

X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)
feature_names = numeric_features + list(preprocessor.named_transformers_['cat'].get_feature_names_ou
```

0.13 5. - 9. OLS Model Training and Evaluation

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import statsmodels.api as sm

X_train_sm = sm.add_constant(X_train_processed)
OLS_model = sm.OLS(y_train, X_train_sm).fit()

print("--- Q2.6: OLS Regression Summary (Statsmodels) ---")
print(OLS_model.summary())

X_test_sm = sm.add_constant(X_test_processed)
y_pred_test = OLS_model.predict(X_test_sm)
```

```
rmse = np.sqrt(mean_squared_error(y_test, y_pred_test))
mae = mean_absolute_error(y_test, y_pred_test)
r2 = OLS_model.rsquared

print("\n--- Q2.8: Evaluation Metrics ---")
print(f"RMSE: {rmse:.2f}")
print(f"MAE: {mae:.2f}")
print(f"R^2 (Train): {OLS_model.rsquared:.4f}")

residuals = y_test - y_pred_test
# Q2.9 Plots must be generated manually
```

0.14 10. Compute Fairness Metrics

```
import seaborn as sns
from scipy import stats

median_salary = y_test.median()
y_pred_binary = (y_pred_test > median_salary).astype(int)

temp_df = pd.DataFrame(X_test_processed, columns=feature_names, index=X_test.index)
temp_df['Actual_Gender'] = X_test['gender']
temp_df['y_pred'] = y_pred_test
temp_df['y_test'] = y_test
temp_df['Residuals'] = residuals
temp_df['y_pred_binary'] = y_pred_binary

group_male = temp_df[temp_df['Actual_Gender'] == 'Male']
group_female = temp_df[temp_df['Actual_Gender'] == 'Female']

print("\n--- Q2.10: Fairness Metrics (Male vs Female) ---")

ms_diff_mf = group_male['y_pred'].mean() - group_female['y_pred'].mean()
print(f"(a) Mean Salary Prediction Diff (M - F): {ms_diff_mf:.2f}")

mae_male = mean_absolute_error(group_male['y_test'], group_male['y_pred'])
mae_female = mean_absolute_error(group_female['y_test'], group_female['y_pred'])
print(f"(b) MAE Male: {mae_male:.2f}, MAE Female: {mae_female:.2f}")

p_male_fav = group_male['y_pred_binary'].mean()
p_female_fav = group_female['y_pred_binary'].mean()
dpd_mf = p_male_fav - p_female_fav
print(f"(c) Demographic Parity Diff (M-F): {dpd_mf:.4f}")

dir_mf = p_female_fav / p_male_fav if p_male_fav > 0 else np.nan
print(f"(f) Disparate Impact Ratio (F/M): {dir_mf:.4f}")

plt.figure(figsize=(10, 6))
sns.violinplot(x='Actual_Gender', y='Residuals', data=temp_df)
plt.hlines(0, -0.5, 2.5, color='red', linestyle='--')
plt.title('Q2.10: Residual Distribution by Gender')
plt.savefig('Q2_10_Residuals_Gender.png')
plt.show()
```

0.15 11. Conduct a statistical test

```
res_male = group_male['Residuals']
res_female = group_female['Residuals']
```

```
t_stat, p_value = stats.ttest_ind(res_male, res_female, equal_var=False)

print("\n--- Q2.11: T-test for Mean Residuals (Male vs Female) ---")
print(f"T-statistic: {t_stat:.3f}, P-value: {p_value:.5f}")
if p_value < 0.05:
    print("Conclusion: Reject H0. The mean residuals differ significantly.")
else:
    print("Conclusion: Fail to Reject H0. No significant difference in mean residuals.")
```

0.16 12. Identify overestimates or underestimates

- **Systematic Overestimation:** Mean(Residuals) < 0.
- **Systematic Underestimation:** Mean(Residuals) > 0.

0.17 13. Use SHAP values

```
import shap

# explainer = shap.Explainer(OLS_model.predict, X_test_sm)
# shap_values = explainer(X_test_sm)

# shap.summary_plot(shap_values, X_test_sm, feature_names=feature_names, show=False)
# plt.savefig('Q2_13_SHAP_Summary.png')

# top_features_indices = np.argsort(np.abs(shap_values.values).mean(0))[::-1][:4][1:]
# for i in top_features_indices:
#     shap.dependence_plot(i, shap_values.values, X_test_sm, feature_names=feature_names, show=False
#     plt.savefig(f'Q2_13_SHAP_Dependence_{feature_names[i]}.png')
#     plt.show()

print("\nQ2.13: SHAP code structure complete.")
```

Question 3: Deep Neural Network Classifier

0.18 Network Architecture and Class Definition

Architecture: Input (784) → FC(256) → ReLU → FC(128) → ReLU → FC(10) (Logits).

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset

class DummyDataset(Dataset):
    def __init__(self, size=1000, features=784, classes=10):
        self.data = torch.randn(size, features)
        self.labels = torch.randint(0, classes, (size,))
    def __len__(self):
        return len(self.labels)
    def __getitem__(self, idx):
        return self.data[idx], self.labels[idx]

class DigitClassifier(nn.Module):
    def __init__(self):
        super(DigitClassifier, self).__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        return x
```

0.19 Training Loop Implementation

```
model = DigitClassifier()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
epochs = 5

train_loader = DataLoader(DummyDataset(size=5000), batch_size=64, shuffle=True)
val_loader = DataLoader(DummyDataset(size=1000), batch_size=64, shuffle=False)

print("--- Q3: Starting Training Loop ---")

for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    for i, data in enumerate(train_loader):
        inputs, labels = data

        optimizer.zero_grad()

        outputs = model(inputs)

        loss = criterion(outputs, labels)
```

```
        loss.backward()

        optimizer.step()

        running_loss += loss.item()

    model.eval()
    val_loss = 0.0
    with torch.no_grad():
        for data in val_loader:
            inputs, labels = data
            outputs = model(inputs)
            val_loss += criterion(outputs, labels).item()

    print(f'Epoch {epoch + 1}/{epochs}, Train Loss: {running_loss / len(train_loader):.4f}, Val Loss

print("--- Q3: Finished Training ---")
```

0.20 1. Why is ReLU preferred over Sigmoid and Tanh in deep networks?

1. **Mitigates Vanishing Gradient.**
2. **Computational Efficiency.**

0.21 2. Explain the role of PyTorch's autograd engine

- **Role:** Automatic differentiation engine.
- **Computation Graph:** Dynamically builds a DAG during the forward pass.
- **Backpropagation:** Traverses the graph backward, applying the chain rule to compute and accumulate gradients.

0.22 3. Submit the python code

The complete Python code is provided in the 'verbatim' blocks in sections 3.1 and 3.2.